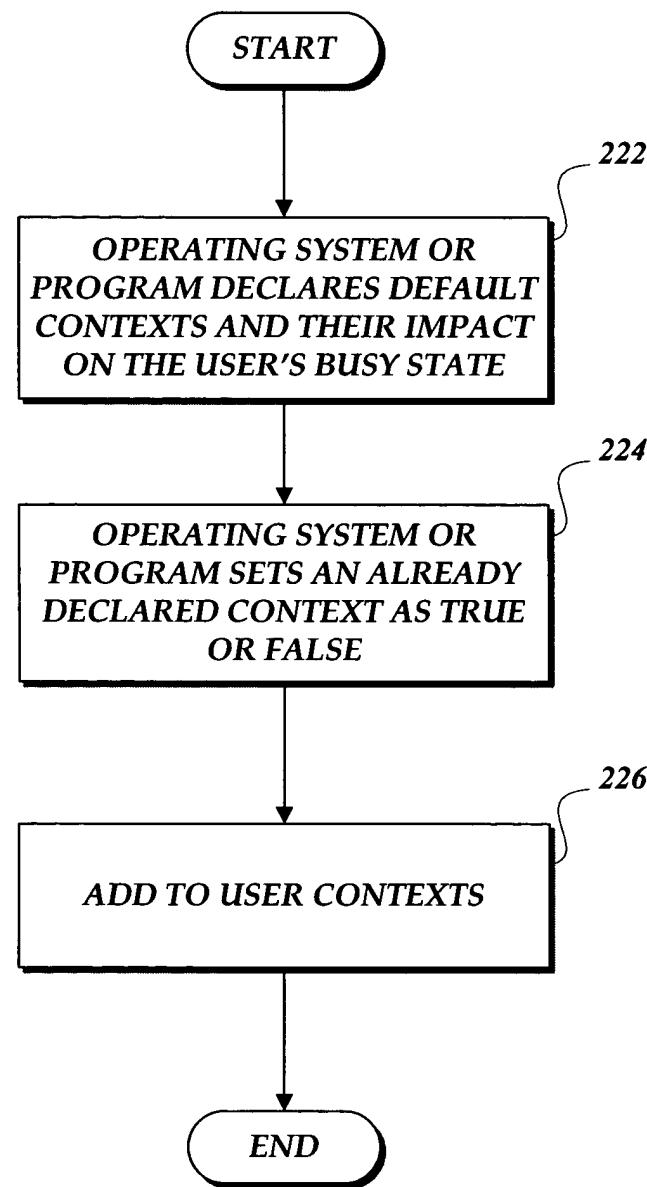


*Fig.2.*

3/25

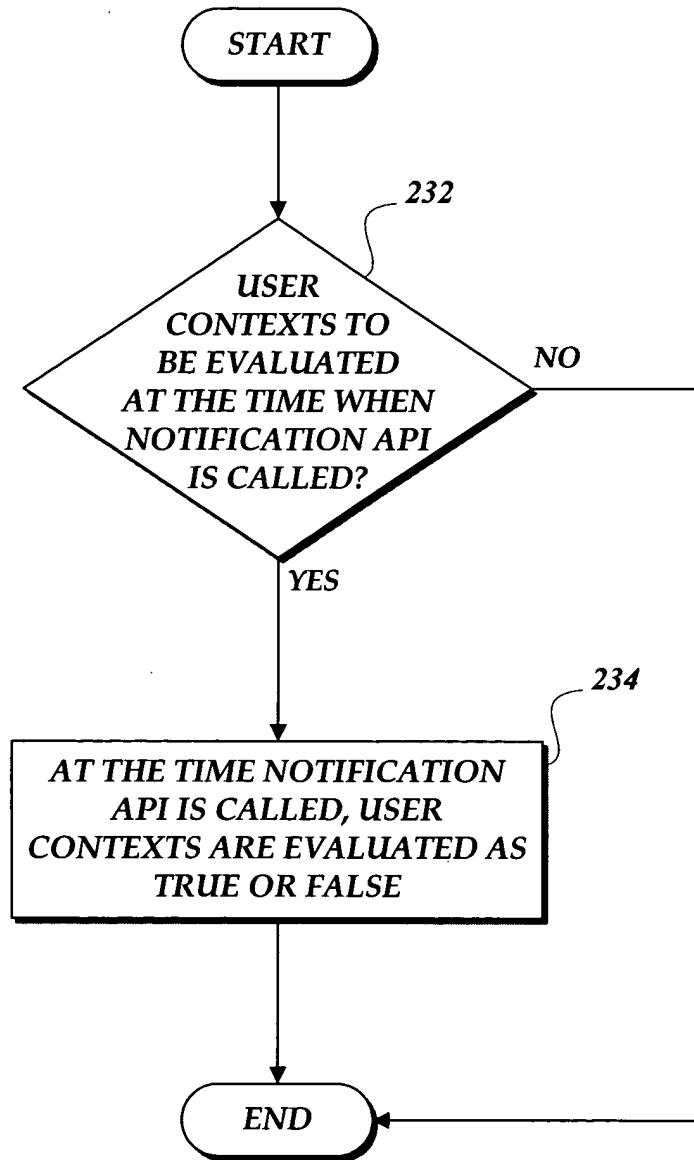
220



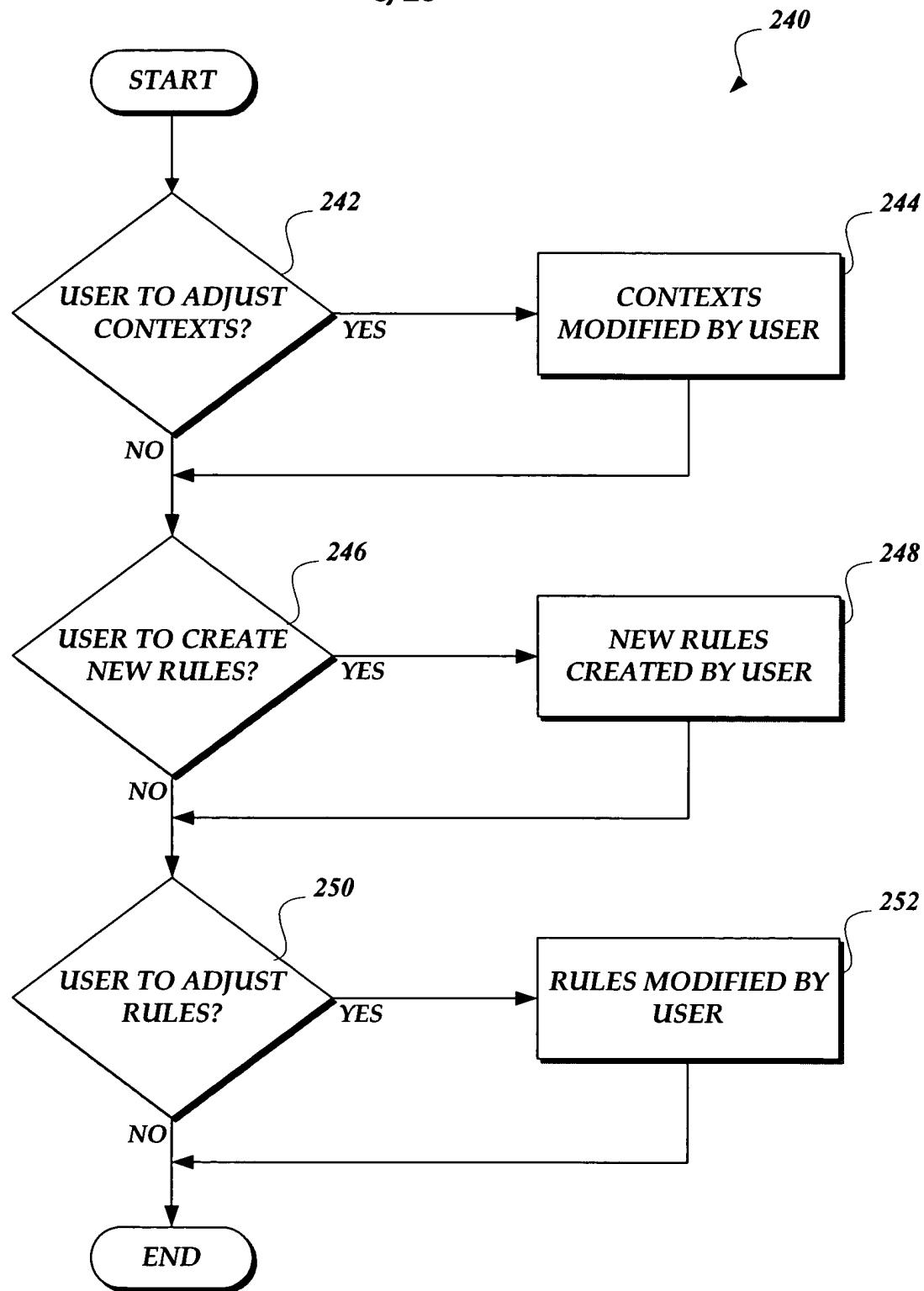
*Fig.3.*

4/25

230

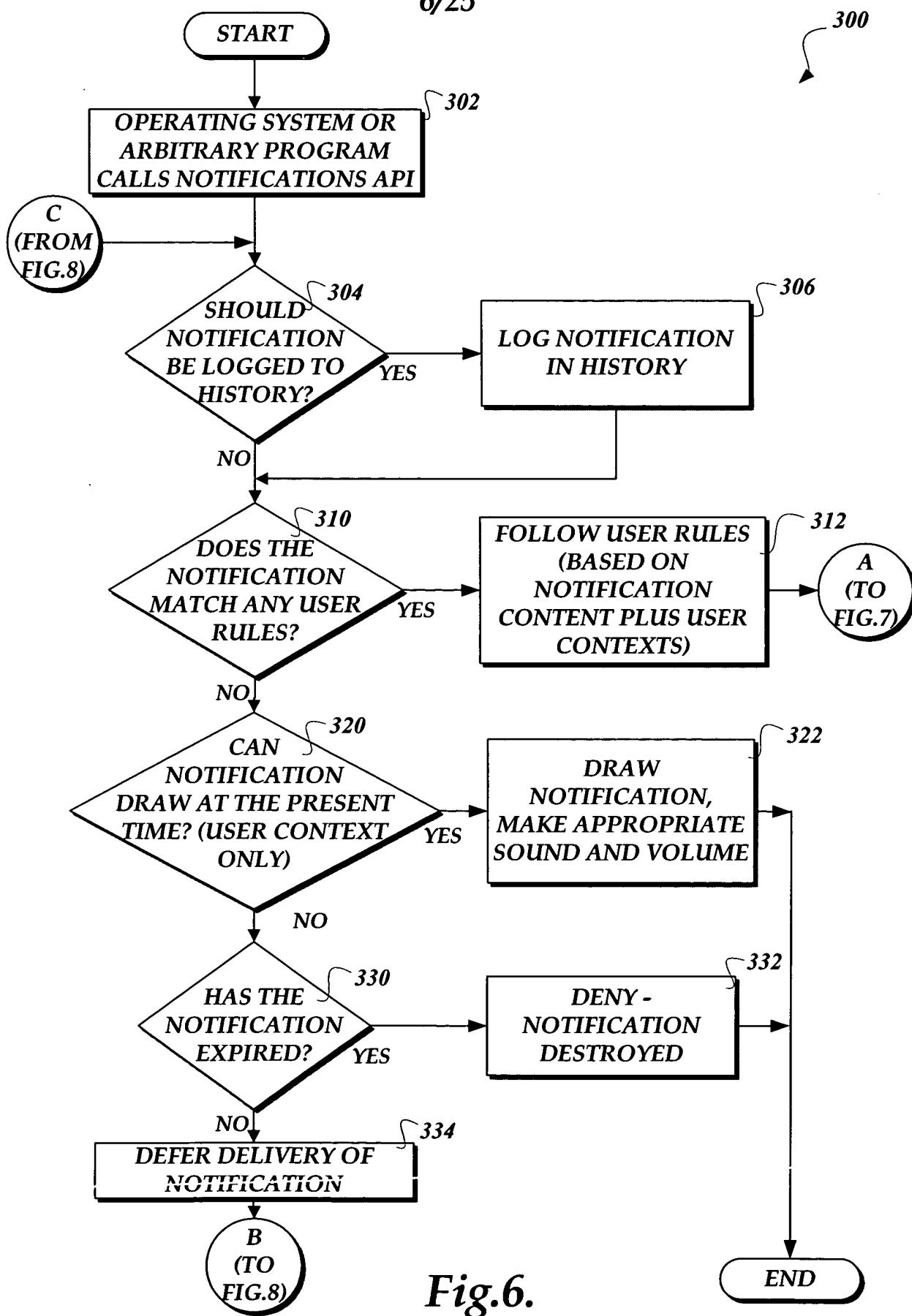


*Fig.4.*



*Fig.5.*

6/25



7/25

350

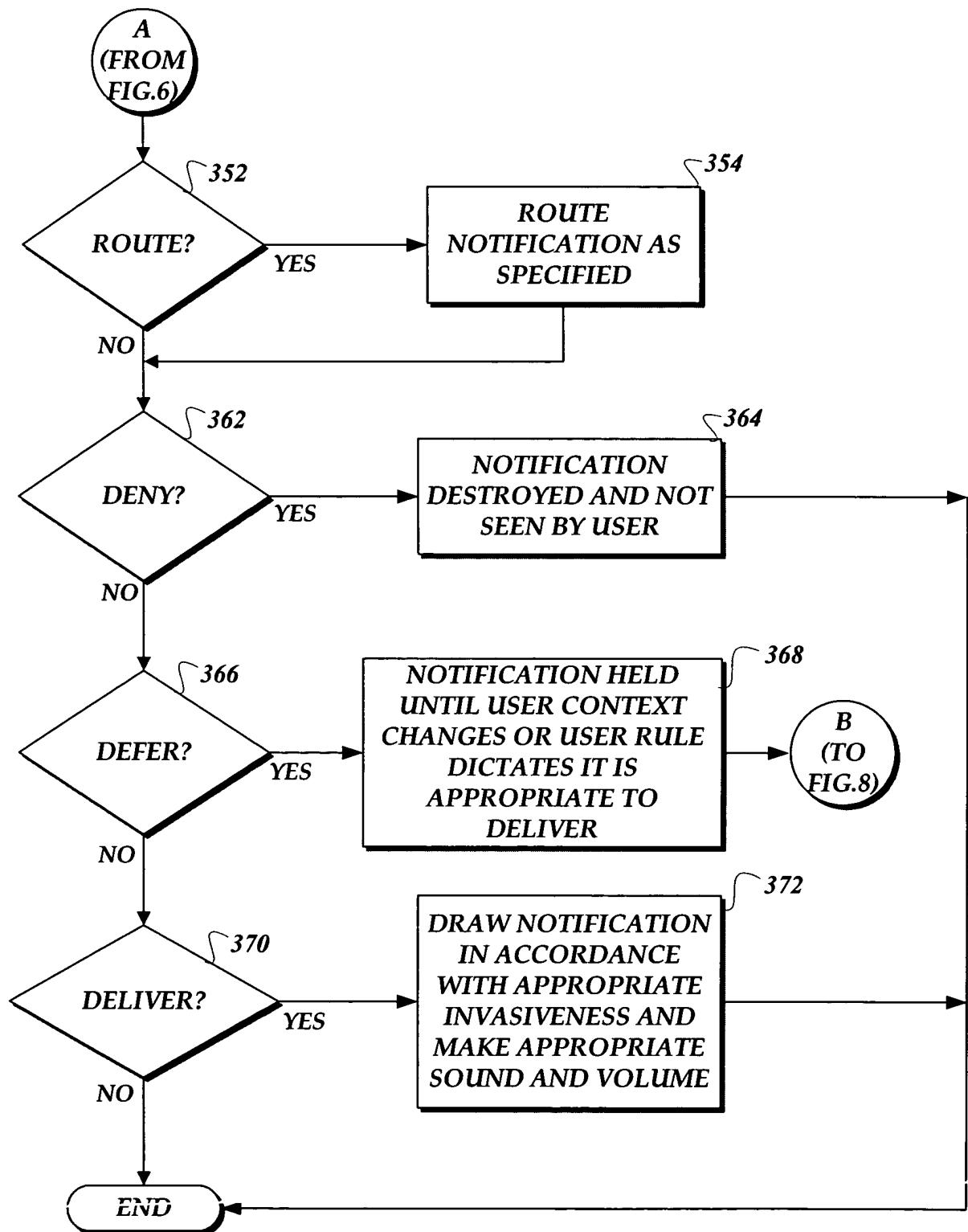
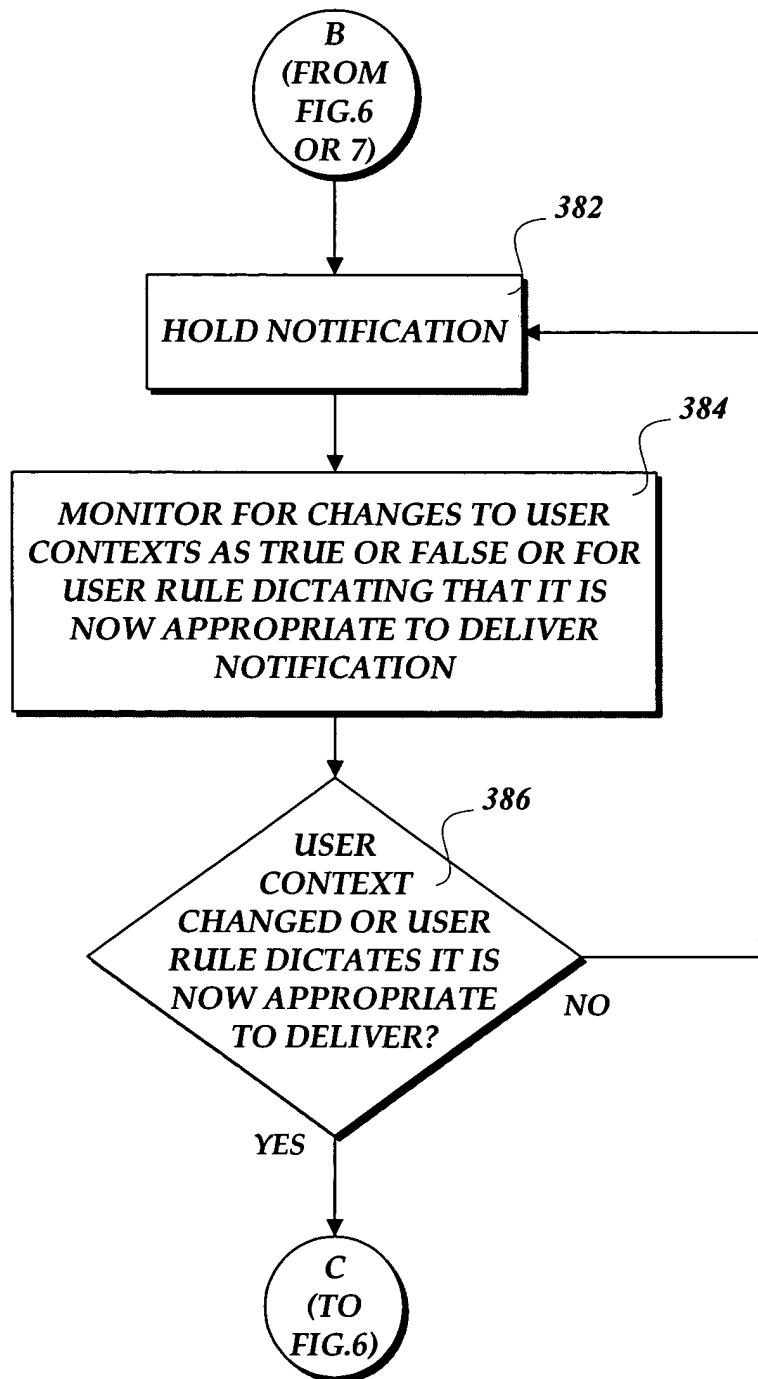


Fig.7.

8/25

380



*Fig.8.*

9/25

400

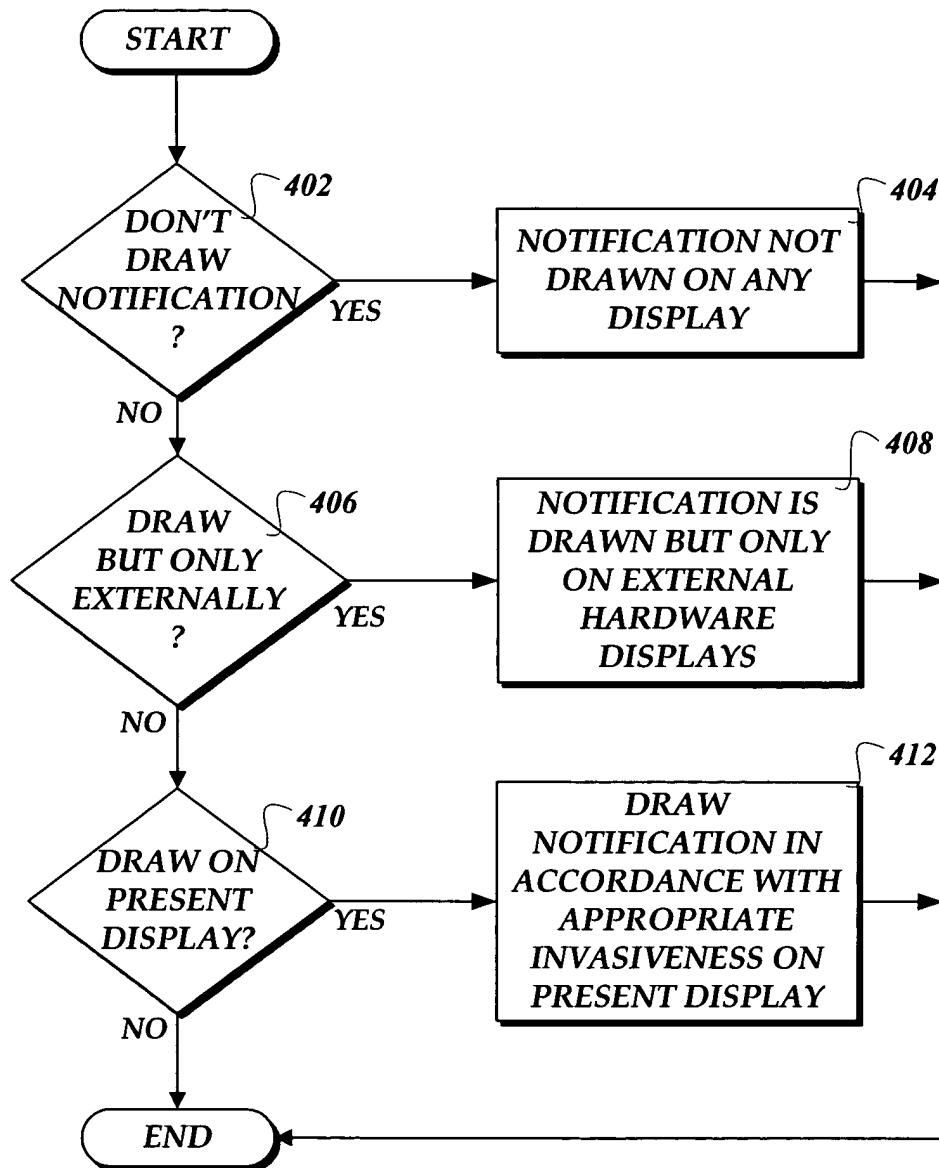


Fig.9.

10/25

420

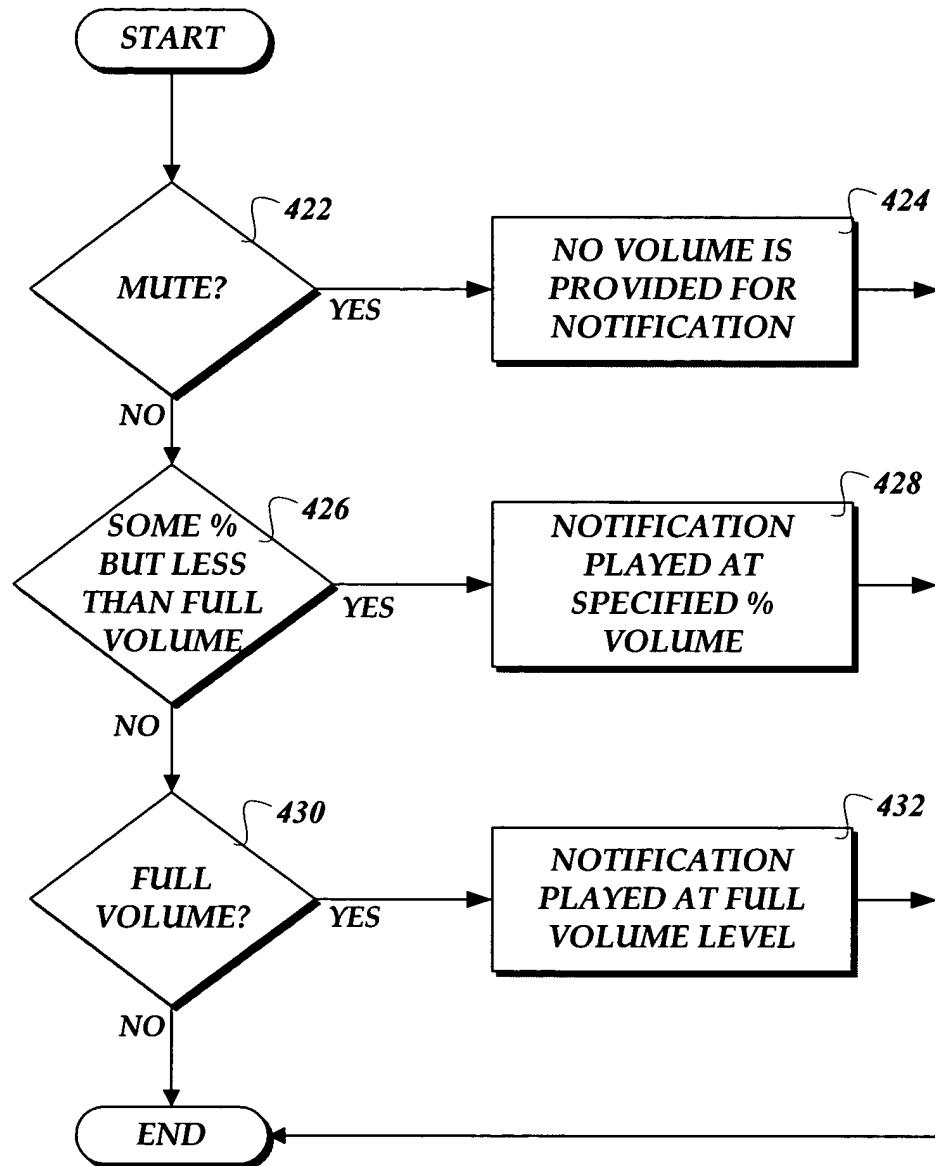


Fig.10.

11/25

500

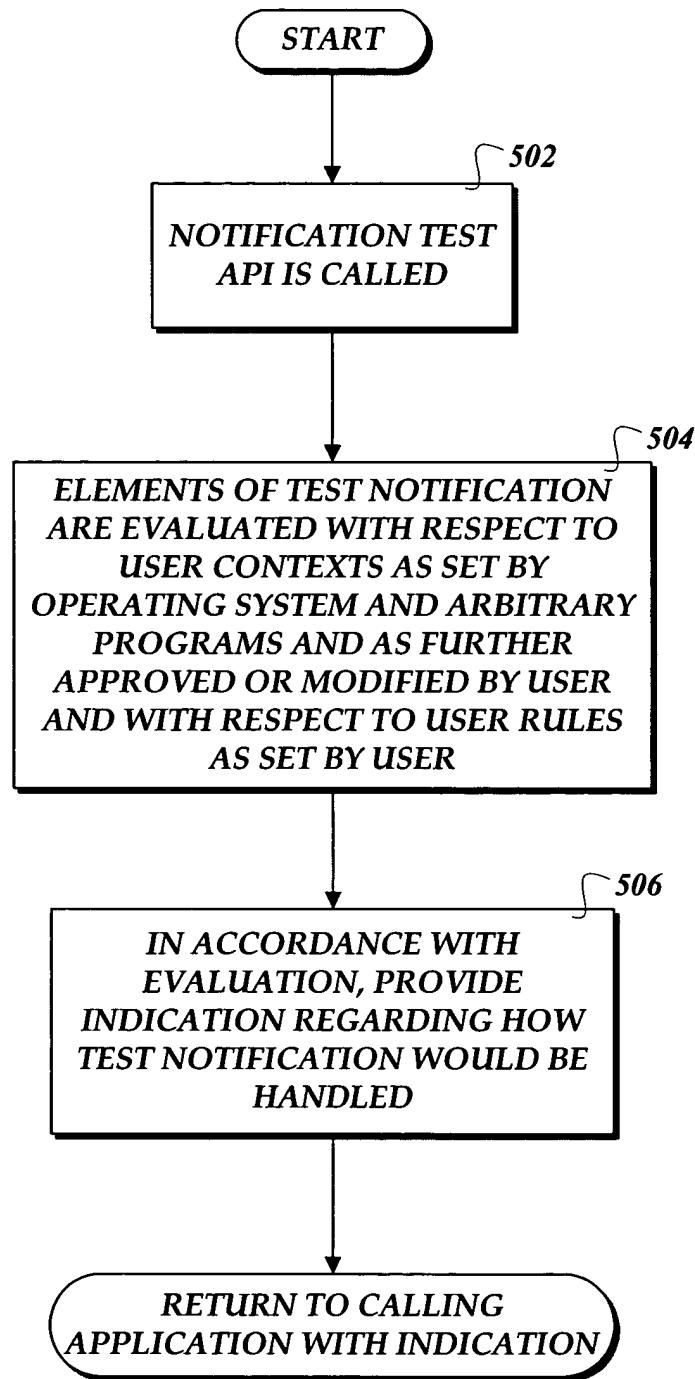


Fig.11.

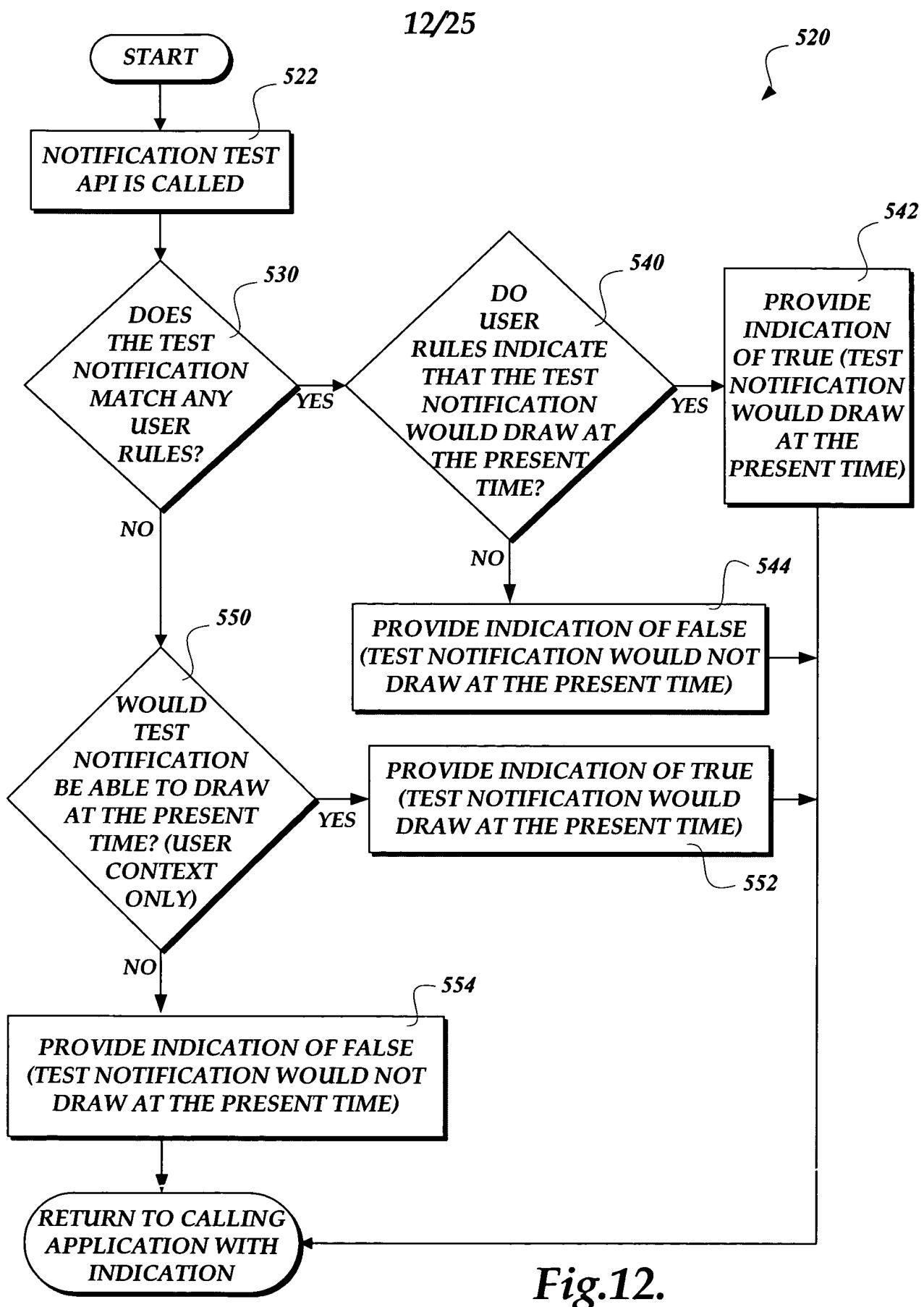
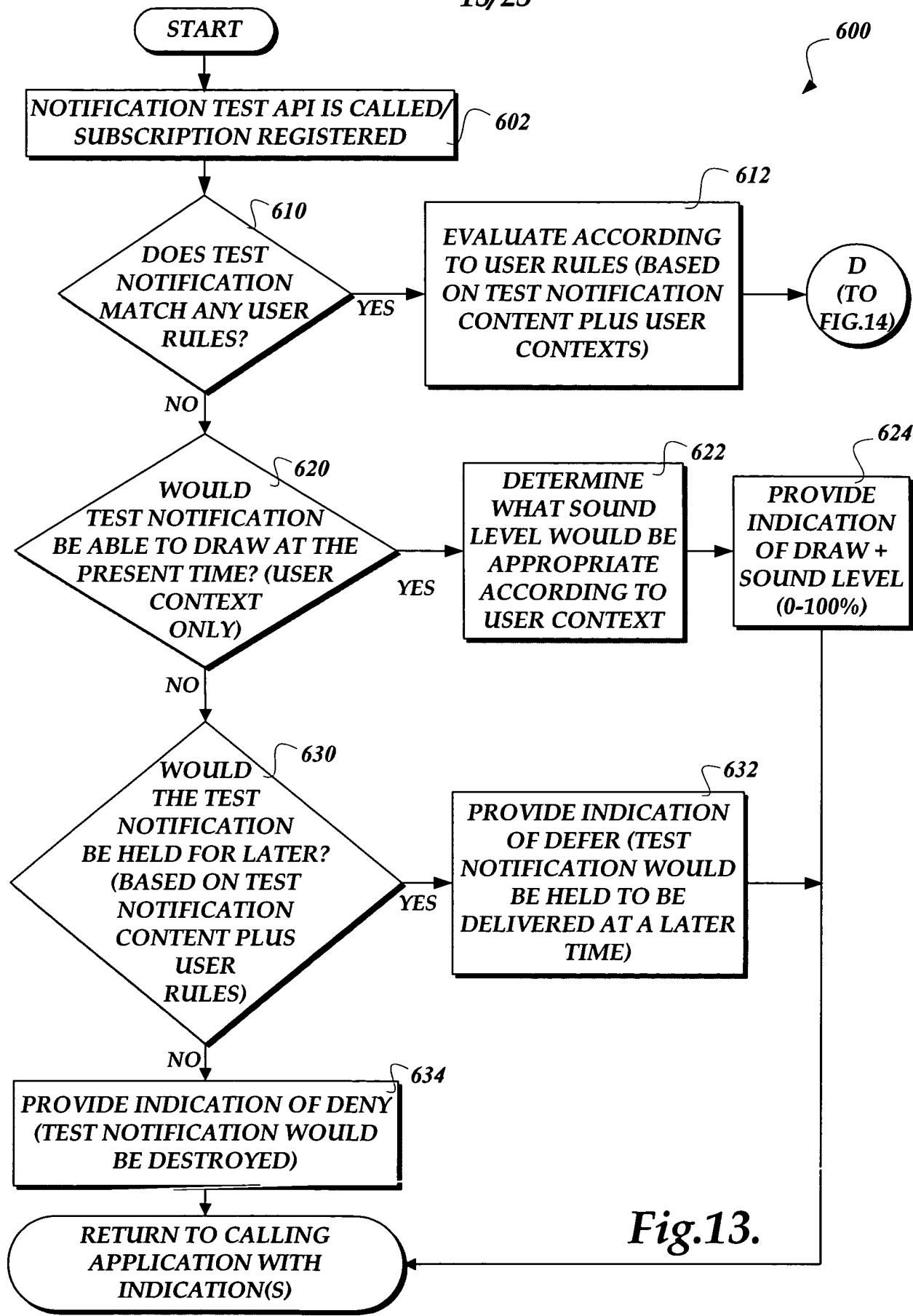


Fig.12.

13/25



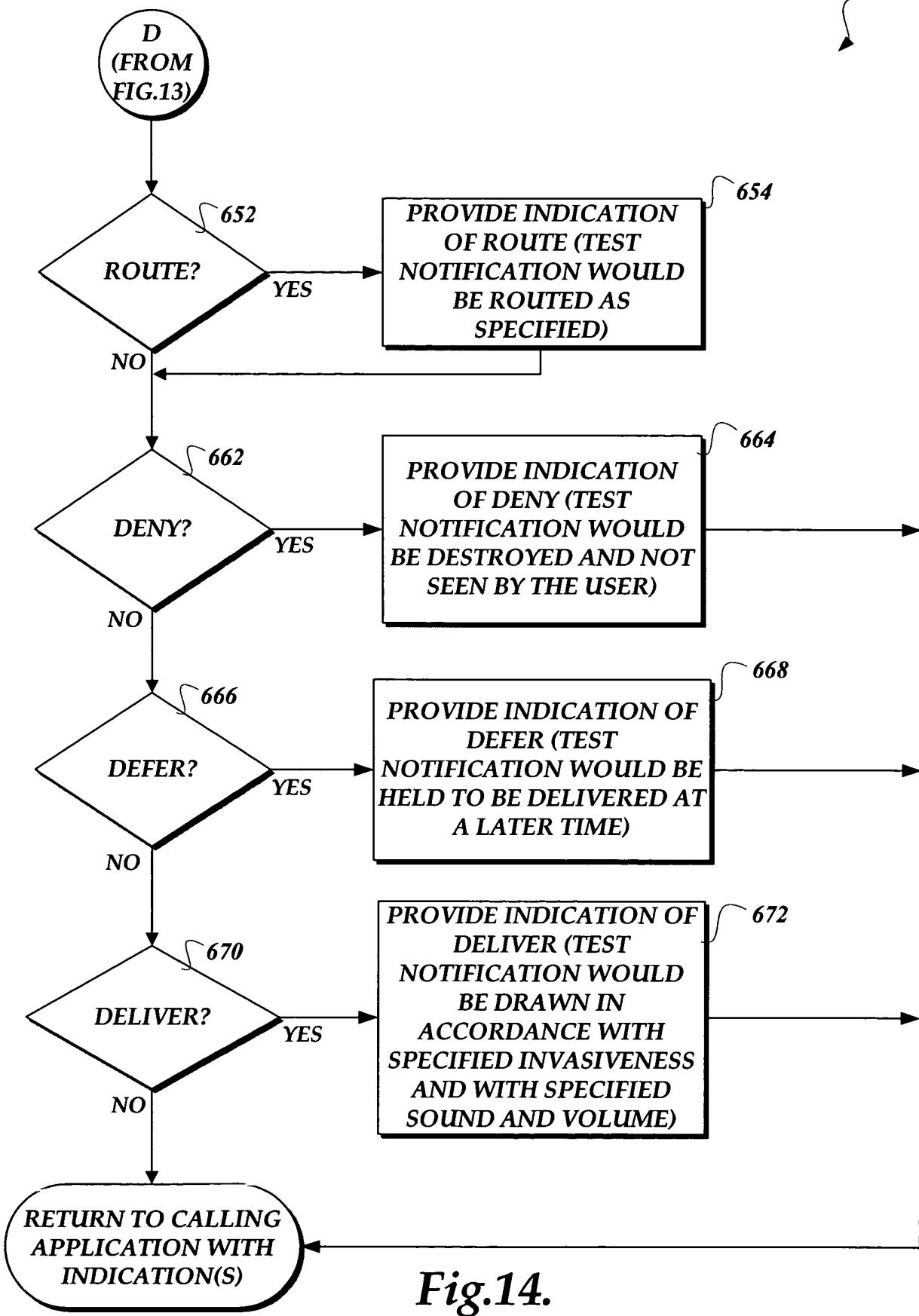


Fig.14.

/// <ExternalAPI>

/// <summary>

/// Use this object to send a balloon Notification to the user. You can derive  
this class if you need to add custom properties to this object so that you can have more  
context when a NotificationClickedEvent arrives.

/// </summary>

public class Notification

{

public:

Notification();

/// <summary>

/// This is the icon for your notification. Minimum size is 16x16, and  
maximum size is 80x80.

/// </summary>

property ImageData HeaderIcon; get, set

/// <summary>

/// This is the title text for your notification.

/// </summary>

property String HeaderText; get, set

/// <summary>

/// This is the main body text of your notification.

/// </summary>

property String BodyText; get, set

/// <summary>

/// Set this property to true if you want the user to be able to click inside a  
notification to trigger some event.

/// </summary>

property bool IsBodyClickable; get, set

void AddButton(NotificationButton button);

event NotificationClickedEventHandler

NotificationClickedEvent;

1510

1520

1530

1540

1550

16/25

1500B

1560

*/// <summary>  
/// Call this method after setting all the relevant Notification properties  
when you are ready to send this so that it will be shown to the user. This method  
returns immediately.*

*/// </summary>  
void Send();*

1570

*/// <summary>  
/// Use this to update a notification that has already been sent. You can  
change some properties and then call Update() to have the changes be reflected.*

*/// </summary>  
void Update();*

1580

*/// <summary>  
/// Call this if you want to know if a particular notification would actually  
draw on screen at this time.*

*/// </summary>  
bool WouldNotificationShow();*

1590

*/// <summary>  
/// Call this when you want to recall a notification that was already sent,  
but that has become obsolete.*

*/// </summary>  
void Revoke();*

*};*

Fig. 15B.

Title: EXTENSIBLE USER CONTEXT SYSTEM FOR  
DELIVERY OF NOTIFICATIONS  
Inventors: T.P. McKee  
Docket No.: MSFT121820

17/25

1600

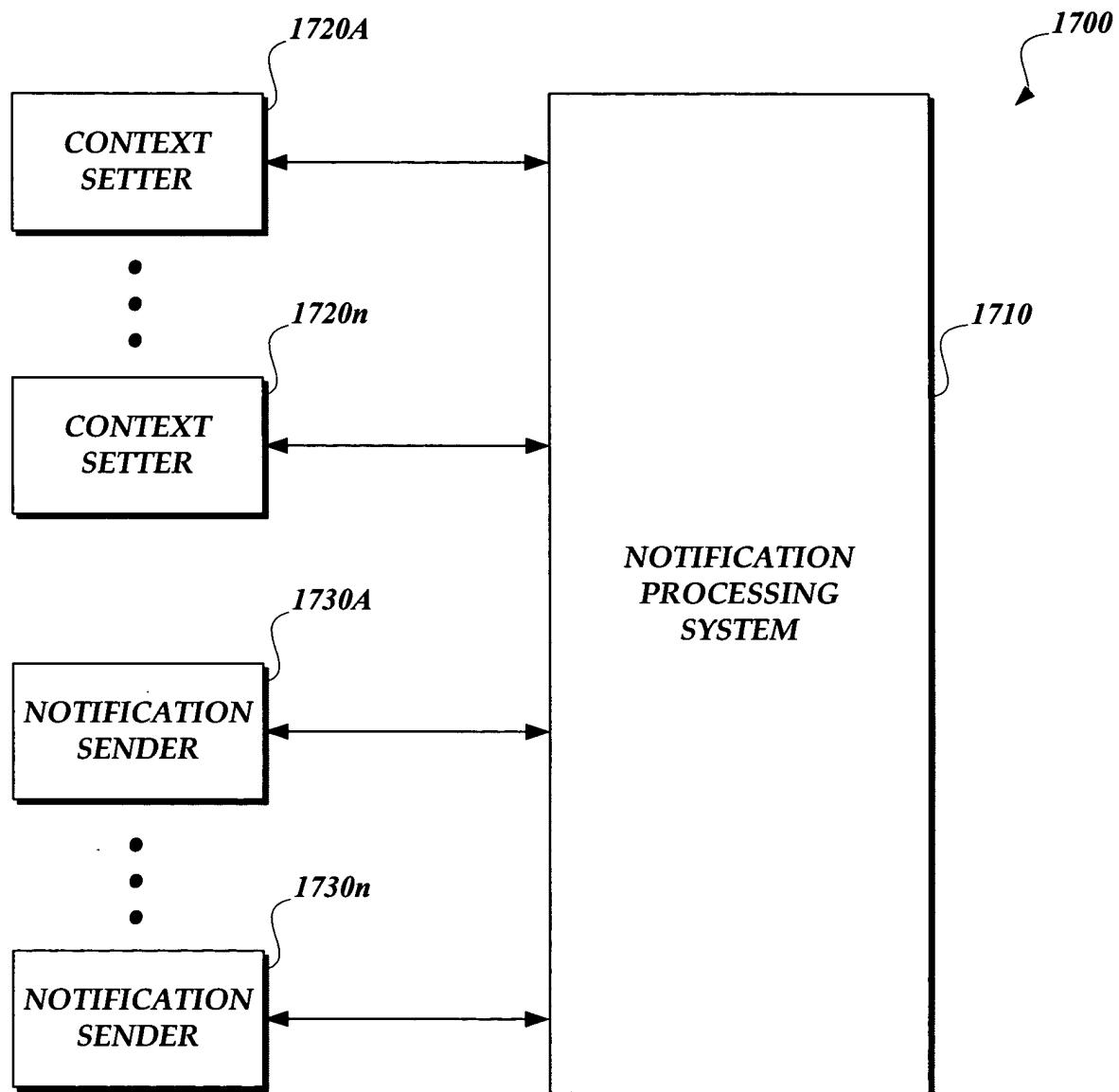
```
/// <ExternalAPI>
/// <summary>
/// This class helps you manage your context changes. Your custom contexts
are defined in your application manifest. This class lets you set your context to true, or
reset it to false. An example of context is the FullScreen context (that context is
managed by the operating system) which is true when any application is full screen.
The default value for any context is always false.
/// </summary>
public class NotificationContext
{
    public:
        /// <summary>
        /// Construct a context object for a particular context, identified by its
        Guid.
        /// </summary>
        NotificationContext(Guid ContextGuid);
        /// <summary>
        /// Change the value of the context. All contexts for an application will be
        reset to false when the application terminates or dies.
        /// </summary>
        property bool Context; set only
};
```

1610

1620

1630

Fig. 16.



*Fig. 17.*

19/25

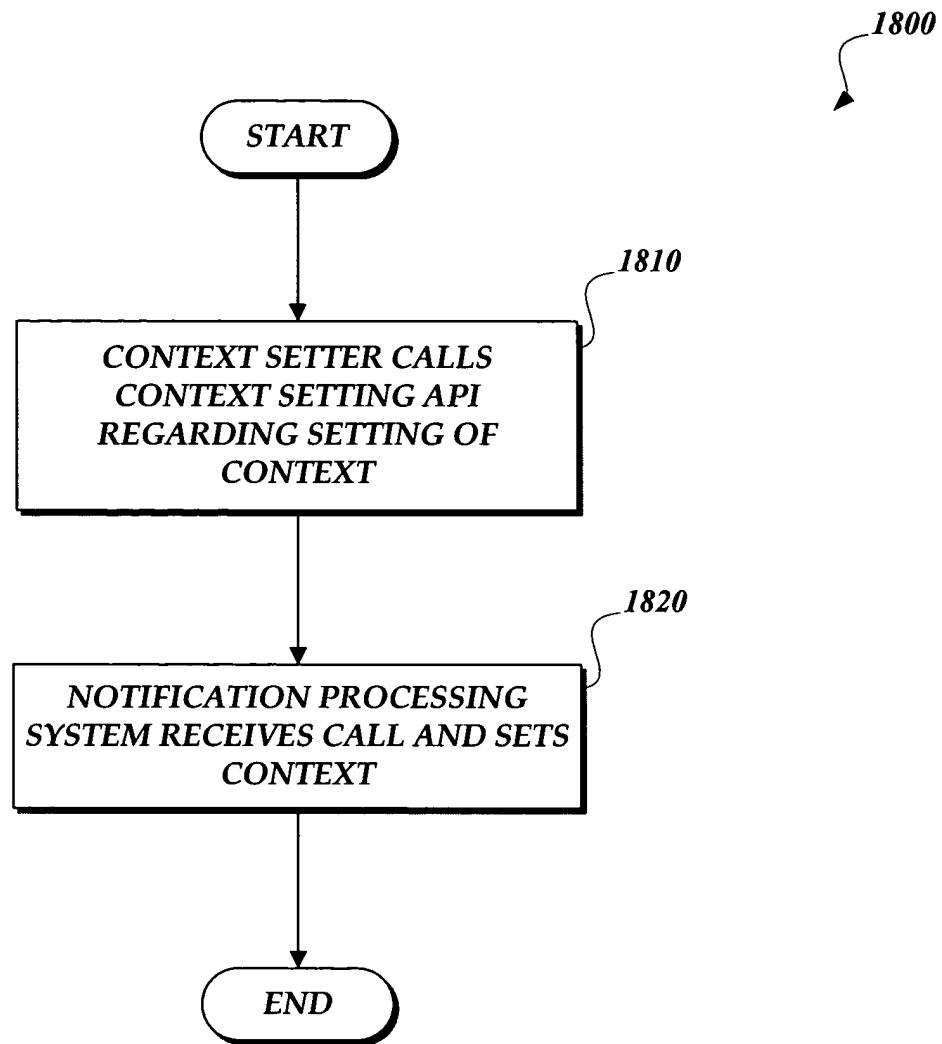


Fig. 18.

20/25

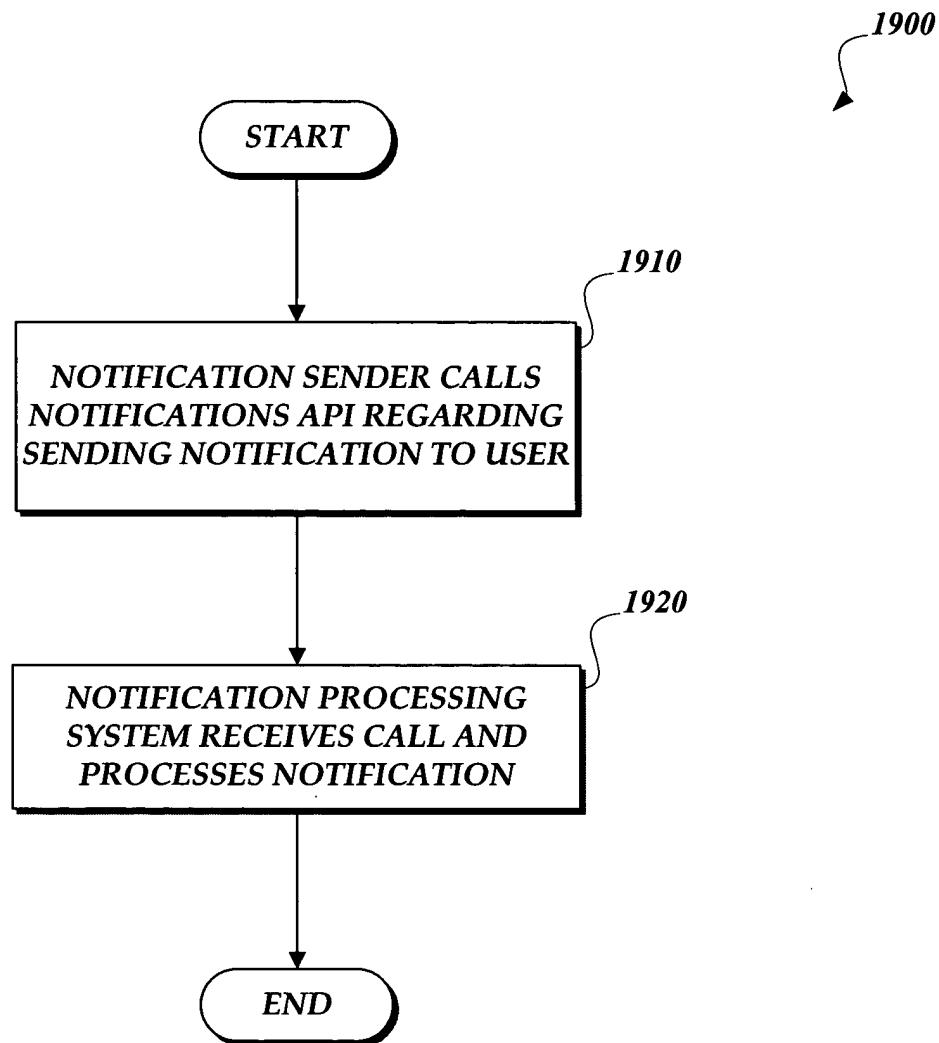
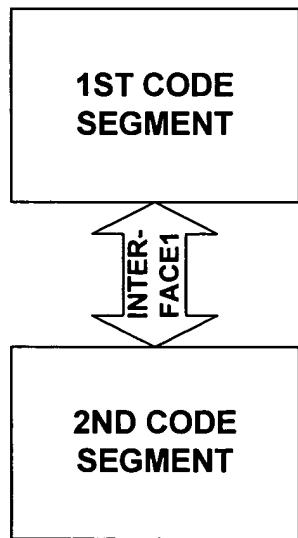
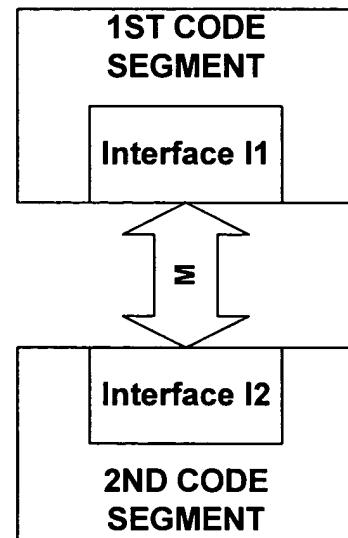


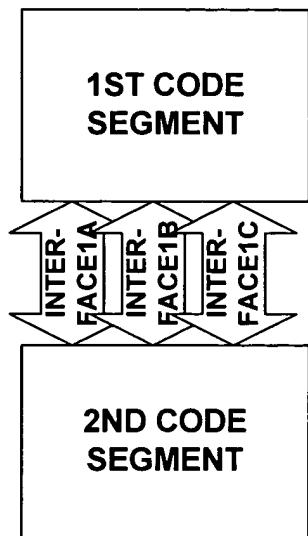
Fig. 19.



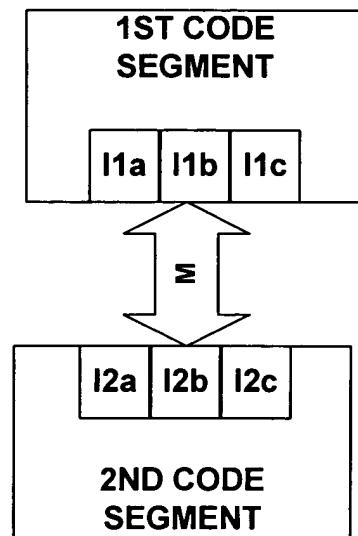
*Fig. 20A*



*Fig. 20B*

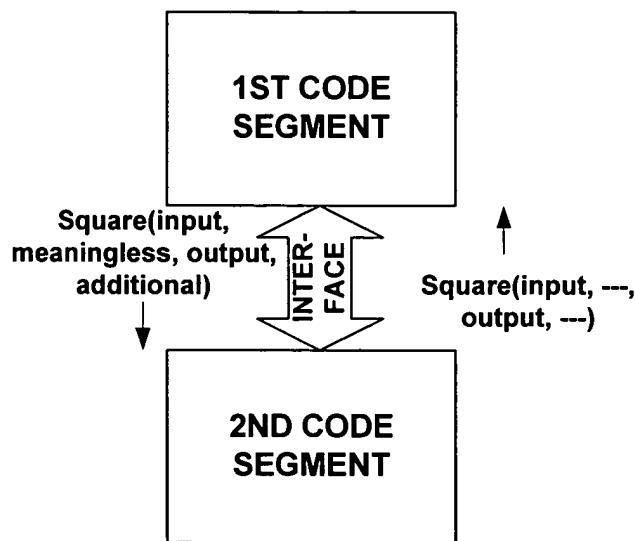


*Fig. 20C*

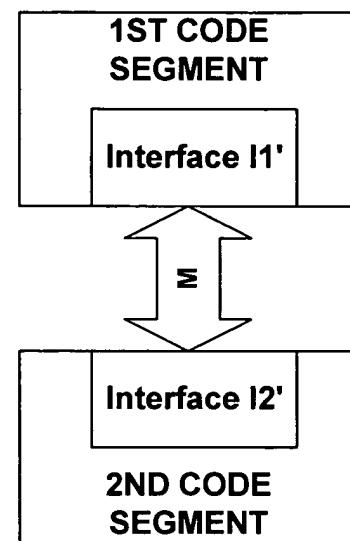


*Fig. 20D*

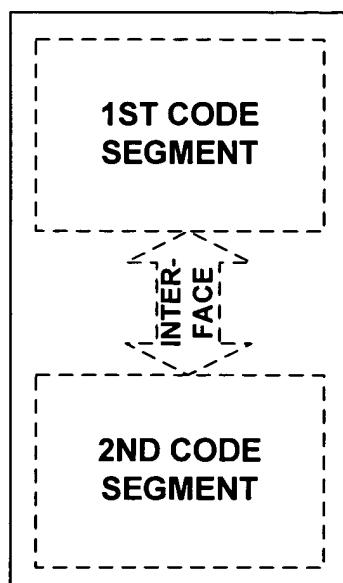
22/25



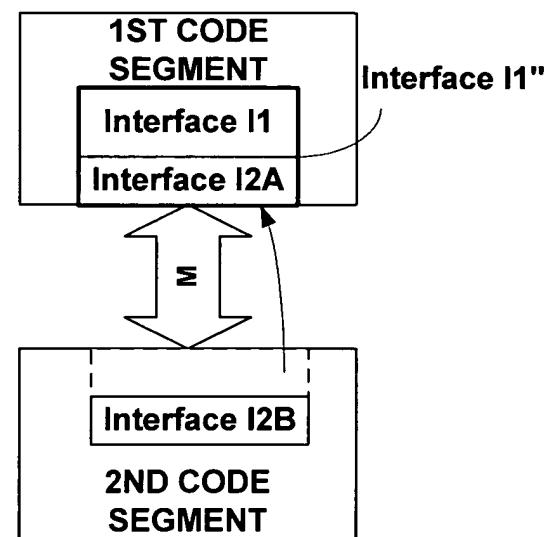
*Fig. 20E*



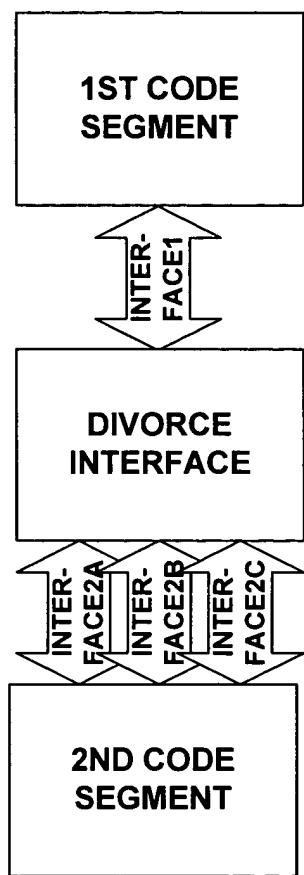
*Fig. 20F*



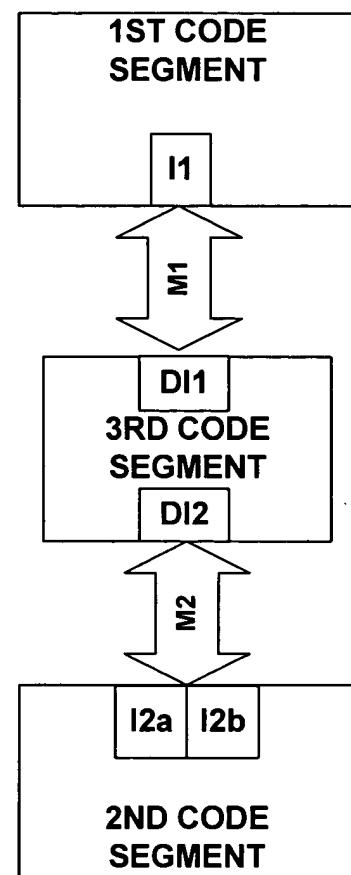
*Fig. 20G*



*Fig. 20H*



*Fig. 20I*



*Fig. 20J*

24/25

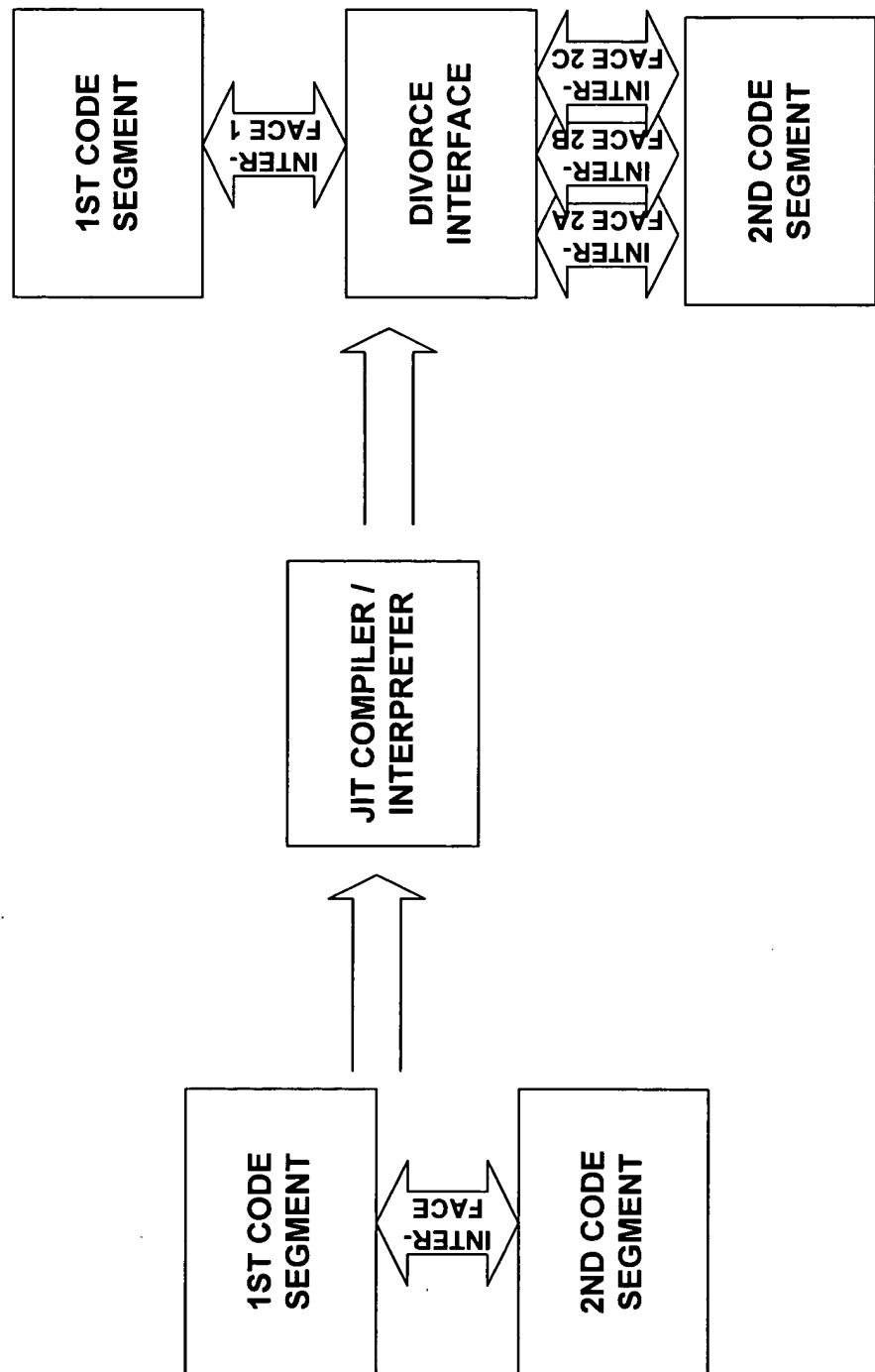
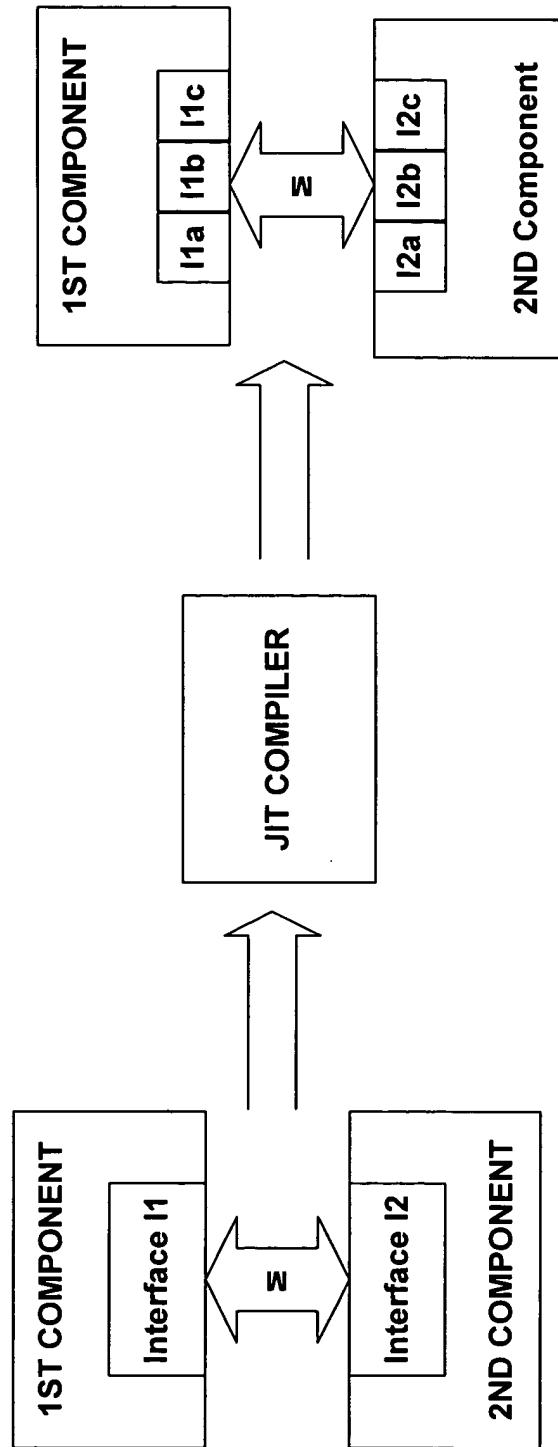


Fig. 20K

25/25



*Fig. 20L*